Introduction

This page briefly describes the algorithm(s) used by MOAB's reader for its native .h5m file format when reading a subset of a file for parallel IO. Please refer to the description of the MOAB/h5m file format before reading this page if you are not already familiar with it.

There are two primary components to reading of a file in parallel in MOAB: 1) specifying the subset of the file to read for each process and 2) Enabling the use of parallel I/O features of the reader for the specific file format. The latter is communicated via specific file reader options and is currently only implemented for the reader for MOAB's native HDF5-based file format (the `ReadHDF5` class.) The subset of the file to read is specified as input to the reader as a list of tags identifying sets or groups of entities in the file to read, and optionally number of parts and part number values indicating to each process which subset of the identified sets to read. This allows for two rather simple mechanisms for specifying the partitioning to the reader for parallel reads. The set or sets identifying the parts of the partition to be used on each processor can either be specified directly via part IDs or the group of all parts of the partition can be identified and the reader instance on each process can be instructed to read an exclusive subset of the part sets by passing the communicator size and rank via the number of parts and part number arguments.

The file subset description is passed to the reader via an instance of the `moab::ReaderIface::SubsetList` struct. This struct contains a list of tag names, each with an optional list of integer tag values, and the struct also contains the optional number of parts and part number arguments. If a tag name is not accompanied by a list of tag values, then it is assumed to mean all sets with the tag defined regardless of the tag values. If a list of tag values is specified, then the tag must contain a single integer value for each entity. If multiple tags are specified, then the group of sets to be read is the intersection of the sets identified by each tag specification. After the list of tagged sets is identified, if the optional number of parts and part number arguments are specified, then each process will read only an exclusive subset of the identified sets. For example, if the communicator size is 10, the part number is 5, and the list of sets identified by the tag list contains 100 sets, then the process will read the 50th through 59th sets in that list, where the sets are in the order that they are stored in the file.

One example of how this data can be used to identify appropriate per-process subsets is to pass the `MATERIAL_SET` tag name with no value list and the communicator size and rank for the number of parts and part number arguments, respectively. Each process will then read an exclusive subset of the material sets (or blocks) from the file such that each material set is read on some process and each process has roughly the same number of material sets. However, which specific material sets are assigned to which process is arbitrary. In the case of the `ReadHDF5` class, they will be assigned based on the order in which they are defined in the file.

As another example, if the mesh was partitioned and the parts were stored as sets tagged with an integer tag named `PARALLEL_PARTITION`, where each such set had a tag value corresponding to the part number, then the parts could be assigned to processors with matching rank by specifying a subset list with a single tag: `PARALLEL_PARTITION` with a tag value of the rank for each process. The number of parts and part number arguments would not be specified in this case.

Assigning parts according to geometric volume ID is an example that would use multiple tag specifications. The subset list would contain the 'GEOMETRIC_DIMENSION' tag with a value of 3 to indicate that we want to specify the file subset using geometric volume sets. The 'GLOBAL_ID' tag would be specified with a value equal to the process rank to indicate that geometric volumes are to be assigned to processes by ID.

Reading File Metadata

A description of the file contents, including flags or values stored in attributes, is read in serial on the root process and broadcast to all other processors. The file is then closed on the root process and re-opened for collective IO on all processes. Once the file is opened for parallel IO no group traversals, search for or reading attributes, etc. is done. The only operations performed in parallel are opening and reading and closing of data sets.

Input Sets

The first step in a partial/parallel read, after the serial read of the file metadata, is to identify which entity sets are to be read based on the input tag specifications. The result of this process will be a list of entity IDs corresponding to the tagged entity sets. The actual data read depends on both the format of the data for the specific tag as stored in the file and whether or not a value list was included with the input specification. If a value list was included with the input spec, then it is necessary to read the tag value table entirely (and collectively) on all processes to identify offsets in the data sets at which matching values are stored. If the tag values are stored in a dense format for entity sets, then that is the only table of tag values read and the entity IDs of the sets are calculated directly from the offsets of the matching values in the data set. If tag values are not stored in a dense format for entity sets, then the sparse data table is read to obtain indices of matching values and the corresponding 'id_list' data set is read (using independent IO) at only those indices to obtain the corresponding entity IDs of the sets. Any entity IDs that do not correspond to entity sets are discarded. The most common scenario will have partition tags stored in the sparse format and no value list specified.

If a fraction of the tagged sets are to be read, then it is only necessary to obtain the list of entity IDs for all of the sets for which the tag is defined. From this, the appropriate subset is selected on each processor as indicated by the input list. If the tag data is stored in the dense format for entity sets then no IO is required. The list of tagged sets can be obtained directly from the file metadata. If the tag data is not stored in the dense format for entity sets, then the `id_list` table must be read completely (and collectively) on all processors to obtain the list of entity IDs for sets for which the tag is defined.

If the input list contains multiple tag specifications, the above procedure is repeated for each specified tag. The final set is the intersection of the results for each tag. The operations described in this section are implemented in the 'moab::ReadHDF5::get_subset_ids' method.

Options Controlling Input Entities

Once the entity sets that define the partition have been identified, it is necessary to gather the list of actual entities to read. In the simplest case this is the set of entities contained directly in the content lists of the partition sets. However, the many practical cases are considerably more complicated. It may be necessary to gather the lists of all recursively contained sets, child sets, etc. and the mesh entities contained in those sets. It may also be necessary to identify lower-dimension side elements of the entities contained in the sets (for example quadrilateral elements that correspond to the sides of hexahedral elements that are to be read.) There are several options that can be specified to control how the reader gathers the final list of entities to read:

- `ELEMENTS={EXPLICIT|NODES|SIDES}`
- `CHILDREN={NONE|SETS|CONTENTS}`
- `SETS={NONE|SETS|CONTENTS}`

The default value for the first option is `SIDES` if the mesh was partitioned based on elements and `NODES` if the vertices were partitioned. The default value for the latter two is `CONTENTS`.

The initial specification of a tag to use to identify set partitions allows the reader to determine an group of sets for which to read the contained entities. The latter two options ('CHILDREN `and` SETS`) determine what is done with sets that are children of the input list or that are contained in the input list. If either option is specified as` NONE `then the corresponding sets are discarded at this stage of the read process. If either option is not specified as` NONE` then the reader will make multiple passes through the set contents and/or child list tables until it has traversed all such links and obtained all sets. If the option is 'CONTENTS', then this recursive read process is done earlier and any mesh elements or vertices contained in the sets are also read.

The `ELEMENTS` option determines how the reader determines which elements beyond those explicitly contained in the above set list are to be read. For a vertex-partitioned mesh, this is typically the only way any elements to be read in are identified. For an element-partitioned mesh, this process is typically used to identify side elements (e.g. explicit quads corresponding to a face of a hex) that are not explicitly contained in the partition sets. If the value is `EXPLICIT` then no additional elements will be read. If the value is `NODES` then the reader will instantiate any element defined in the file for which it already intends to read all of the defining vertices. If the value is `SIDES` then any that correspond to the sides of elements contained in the partition sets will also be read.

Further discussion will assume the default values for these options.

Gathering Input Entities

The `read_set_ids_recursive` subroutine is used to gather contained and child sets from the partition sets. As the partition sets rarely contain other sets or have child sets, this typically results in a pass over the relevant data sets. The function will iteratively alternate between calls to read child IDs and contained IDs for each newly identified list of sets until no new sets are found, beginning with the partition sets. The child IDs are read iteratively for each contiguous block of set file IDs. First either one or two reads are done from the column of the set description dataset to obtain the range of indices into the set child list dataset that must be read. One or more reads into the child list data set are then done (reading as much as possible into a 4 MB buffer) to obtain file ids of child sets. The process of reading contained IDs is similar, but is complicated by the need to read the set type bitfield for each set from the set description dataset to determine if the set contents list is in a ranged format. At this stage, any contained IDs corresponding to non-set entities are discarded.

Once the group of sets containing entities to be read have been identified, the contents of all such sets are read again to obtain the list of file IDs for any elements or vertices that are to be read from the file. The connectivity for all identified elements is then read to determine the file IDs of any vertices which are to be read from the file.

Reading Data

One the list of file IDs for for vertices and mesh elements has been obtained, the reader will read all indicated entities and instantiate them in the MOAB database. Node coordinates are read first. The reader will read each column of the table of node coordinates separately, as this allows a single collective read into MOAB's internal buffers for storing coordinates. Only the coordinates for the desired vertices are read.

Next the `read_elements_and_sides` subroutine is used to read the connectivity of all elements of interest and to instantiate those elements in the MOAB database. This subroutine first identities the maximum topological dimension of the elements to be read. It then iterates over each element group in the file corresponding to elements of a smaller dimension. It collectively reads all of the connectivity data for all such elements on every processor and instantiates any elements it reads for which all of the vertices have been read. The subroutine then reads all explicitly identified elements with the identified maximum topological dimension. As a final step, any of the lower-dimension elements that do not correspond to a side of some explicitly specified element are deleted (e.g. elements spanning a one-element wide gap between partitions.)

At this point another pass over the set contents table is made. This time every processor collectively reads the entire table, searching for any sets that contain any of the elements or vertices read from the file and appending the corresponding file IDs to the list of sets to be instantiated. After that, the list of sets to be instantiated is complete and the actual sets contents, child, and parent lists are read from the file and the sets are instantiated.

The final phase is the read of all tag data for all vertices, elements, and sets read from the file.